RAUC

Behind the Scenes of an Update Framework

ELC Europe 2019

Enrico Jörns – e.joerns@pengutronix.de

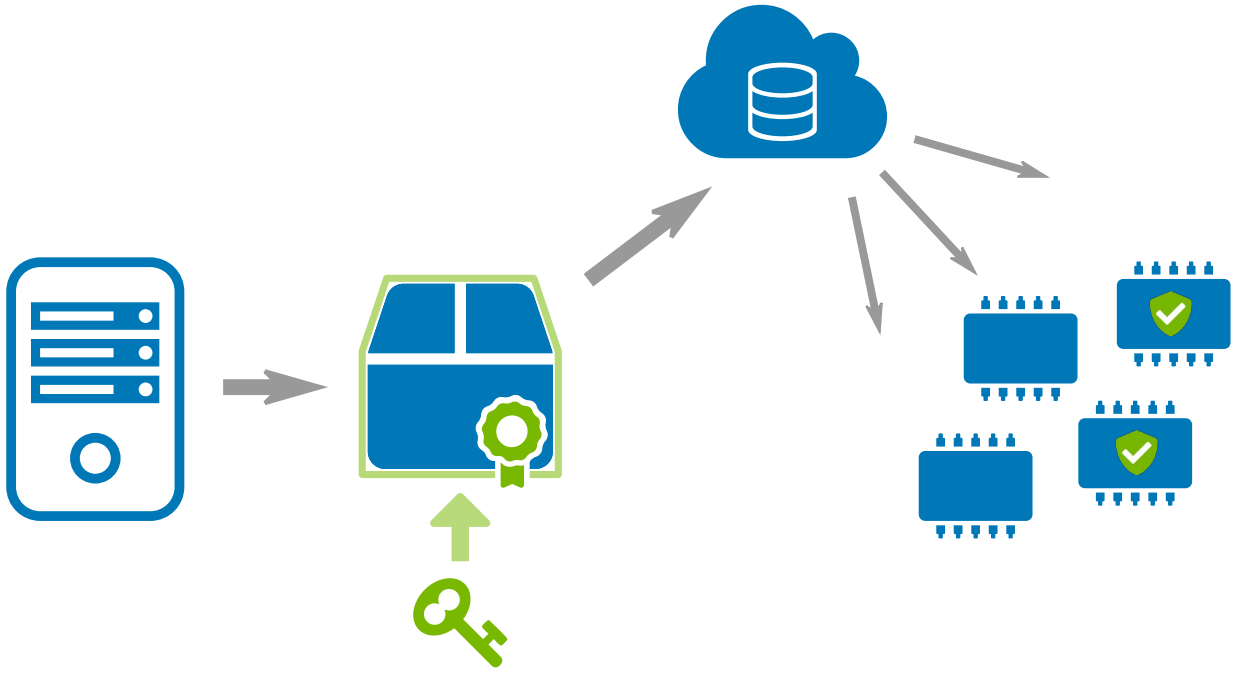
About Me

- Embedded software developer
- RAUC co-maintainer
- At Pengutronix since 2014

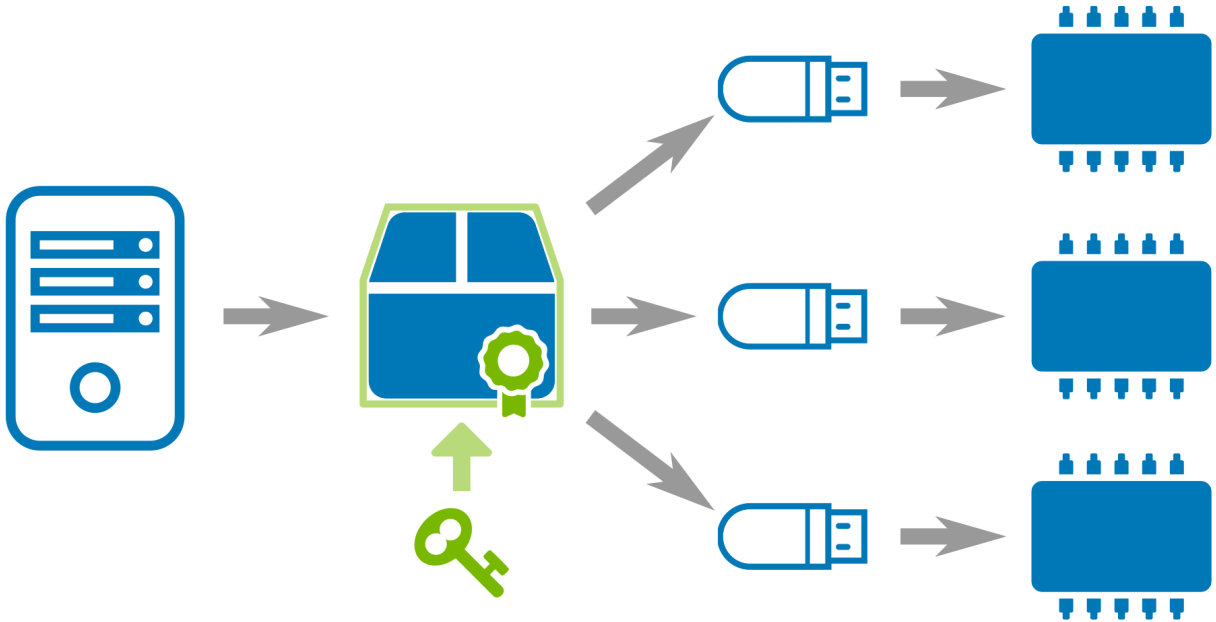
- Embedded Linux consulting & support since 2001
- > 5000 patches in Linux kernel

Introduction

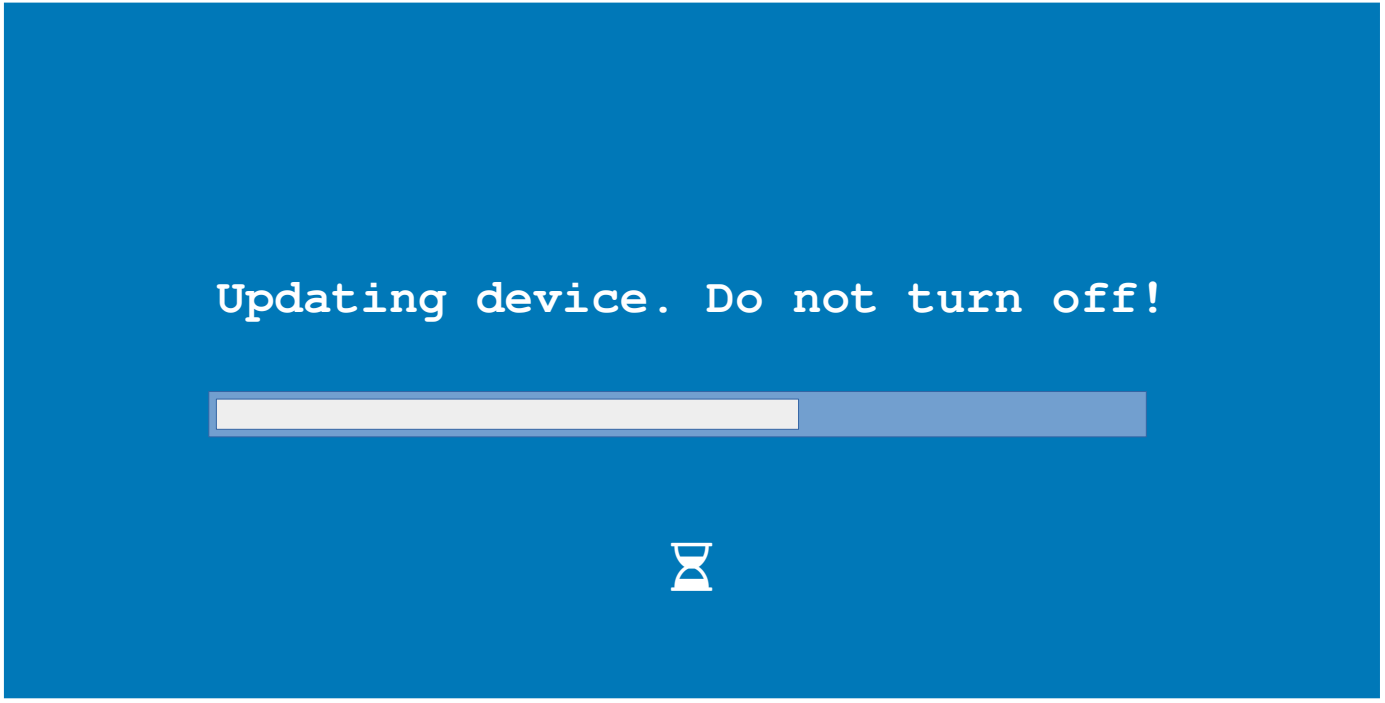
Updating – Big Picture



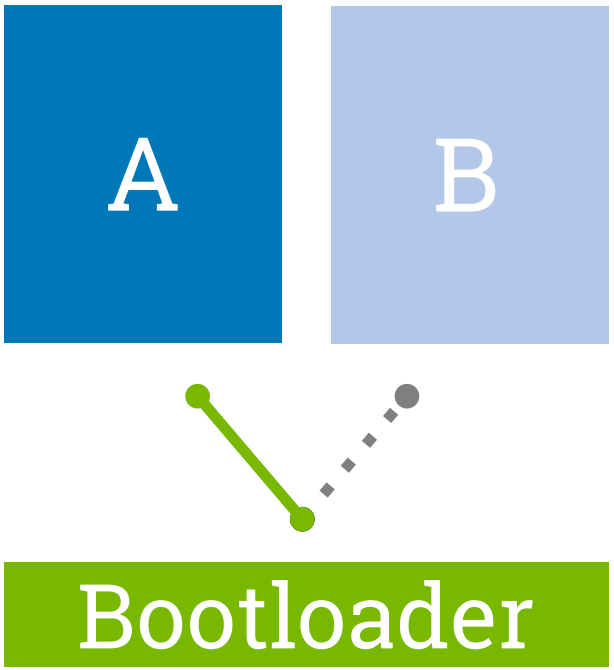
Updating – Big Picture



Fail-Safe Updating – Atomicity + Redundancy



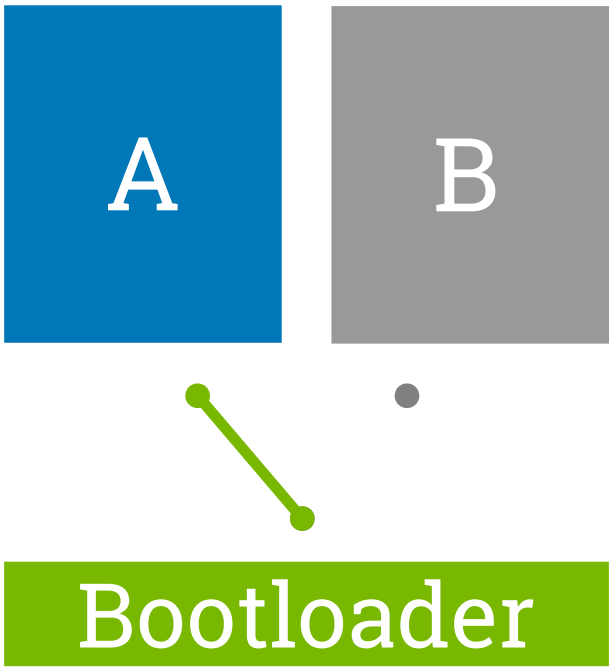
Fail-Safe Updating



A: Active (running) system

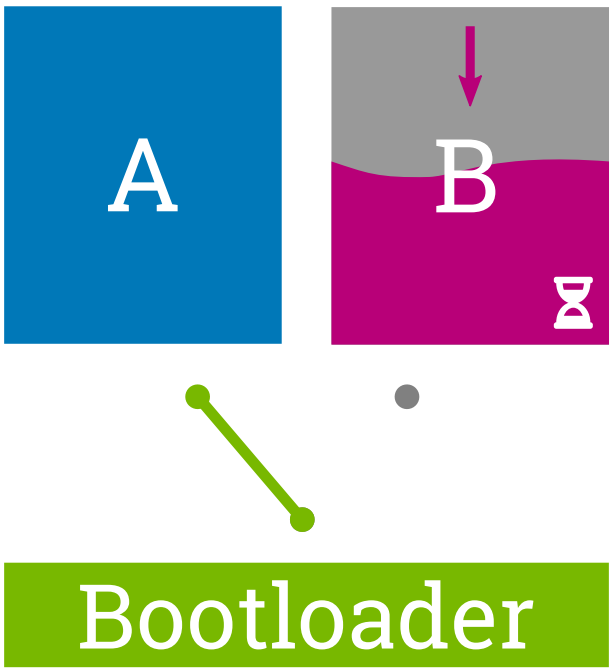
B: Inactive system

Fail-Safe Updating



- Deactivate partition to update

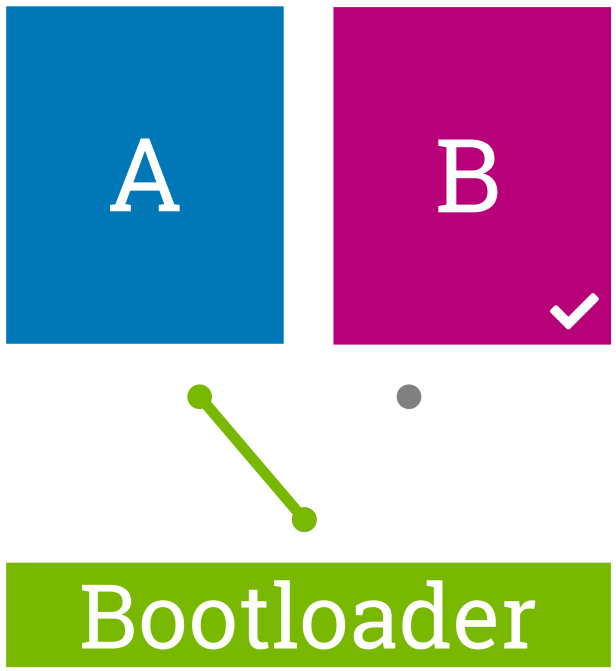
Fail-Safe Updating



- Deactivate partition to update
- Write update(s) to disk

  Critical Operation!

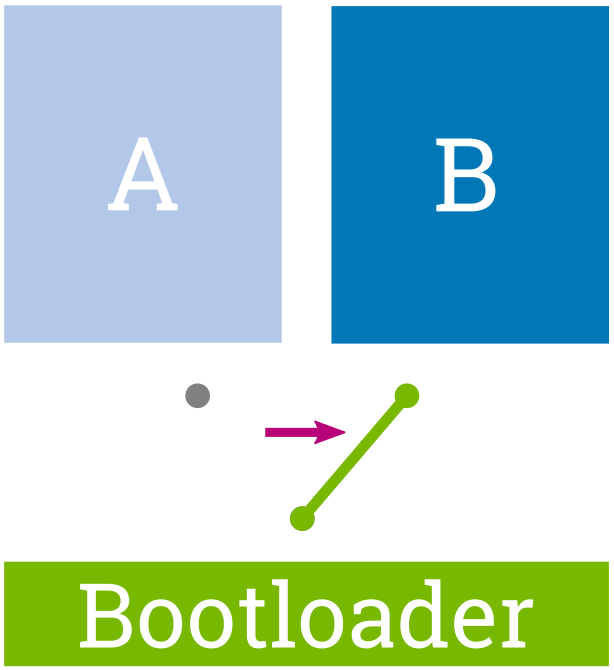
Fail-Safe Updating



- Deactivate partition to update
- Write update(s) to disk

  Critical Operation!
- Update fully completed + verified, etc.

Fail-Safe Updating

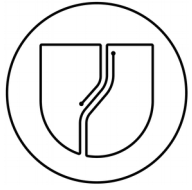


- Deactivate partition to update
- Write update(s) to disk

  Critical Operation!
- Update fully completed + verified, etc.
- Activate updated slot

Overview Over FOSS Update Frameworks

- Image-based



swupdate

- Others

OSTree

RAUC

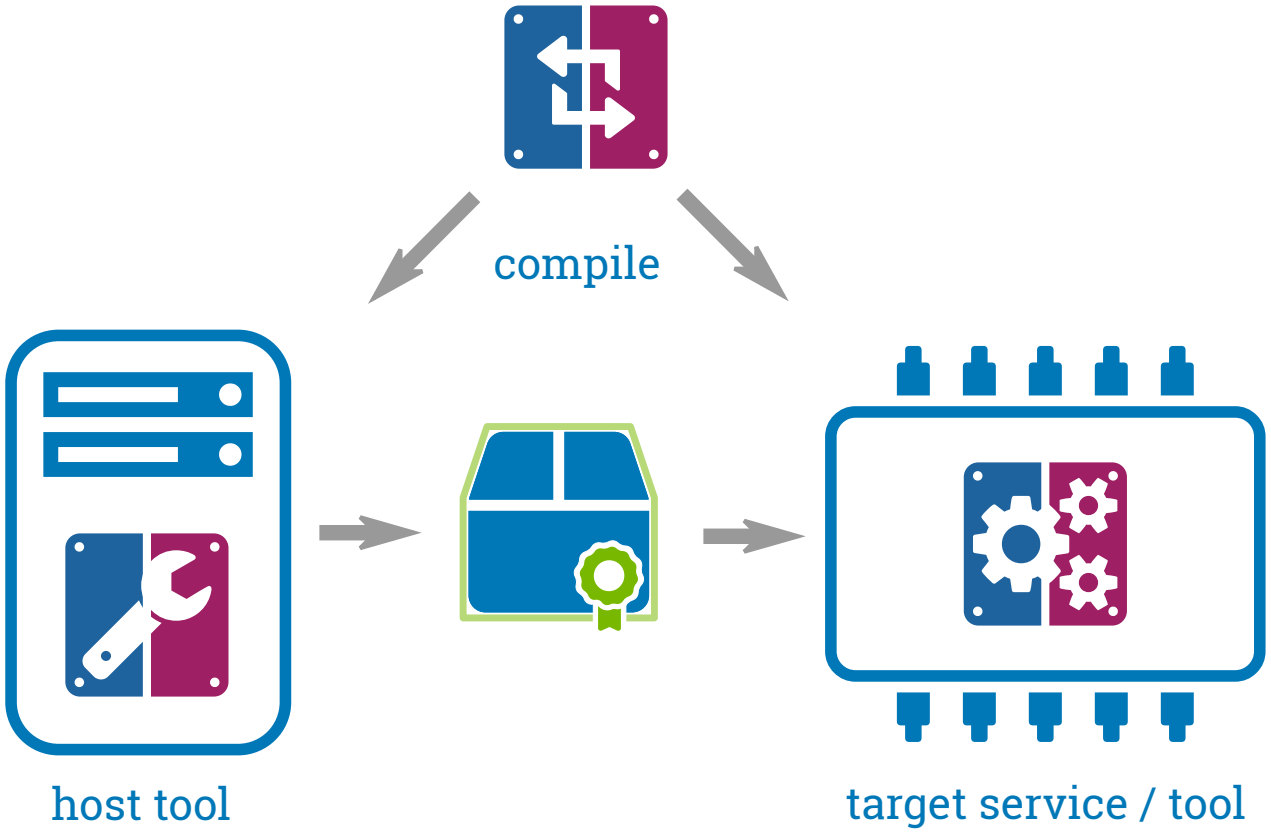
about:rauc

- Name: „Robust Auto-Update Controller“
- Subject: FOSS update framework
- Age: Started in 2015, project-driven
- License: LGPL-2.0
- Version: 1.2
- Community: ~50 contributors (1450 commits)
- Recent: Ongoing development / fixes / adaptions

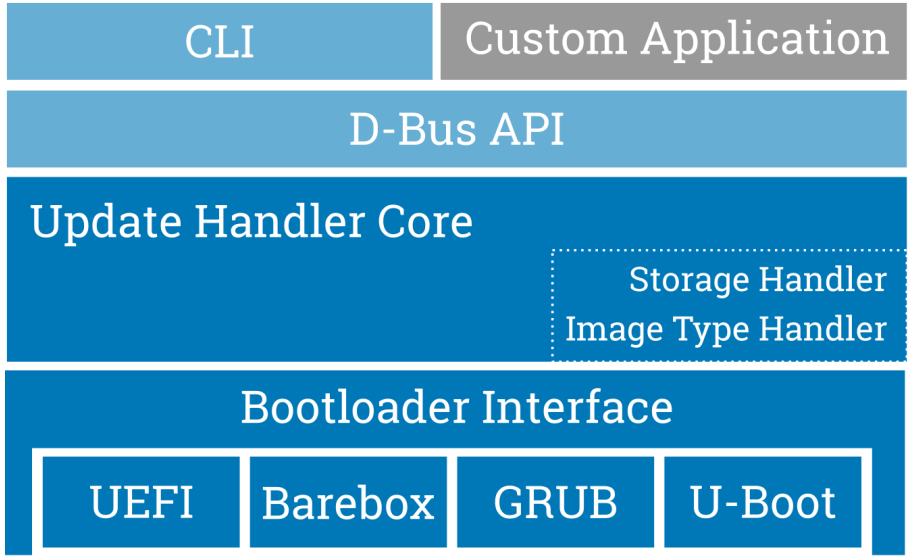
RAUC – Design Goals

- Generic declarative framework
  - A+B, A+recovery, A+B+recovery
  - Platforms / bootloaders
  - Application integration
- Limited complexity
- Security (update verification)
- Robust design
  - Error Handling, standard libaries, subprocess calls

RAUC – Host + Target Tool



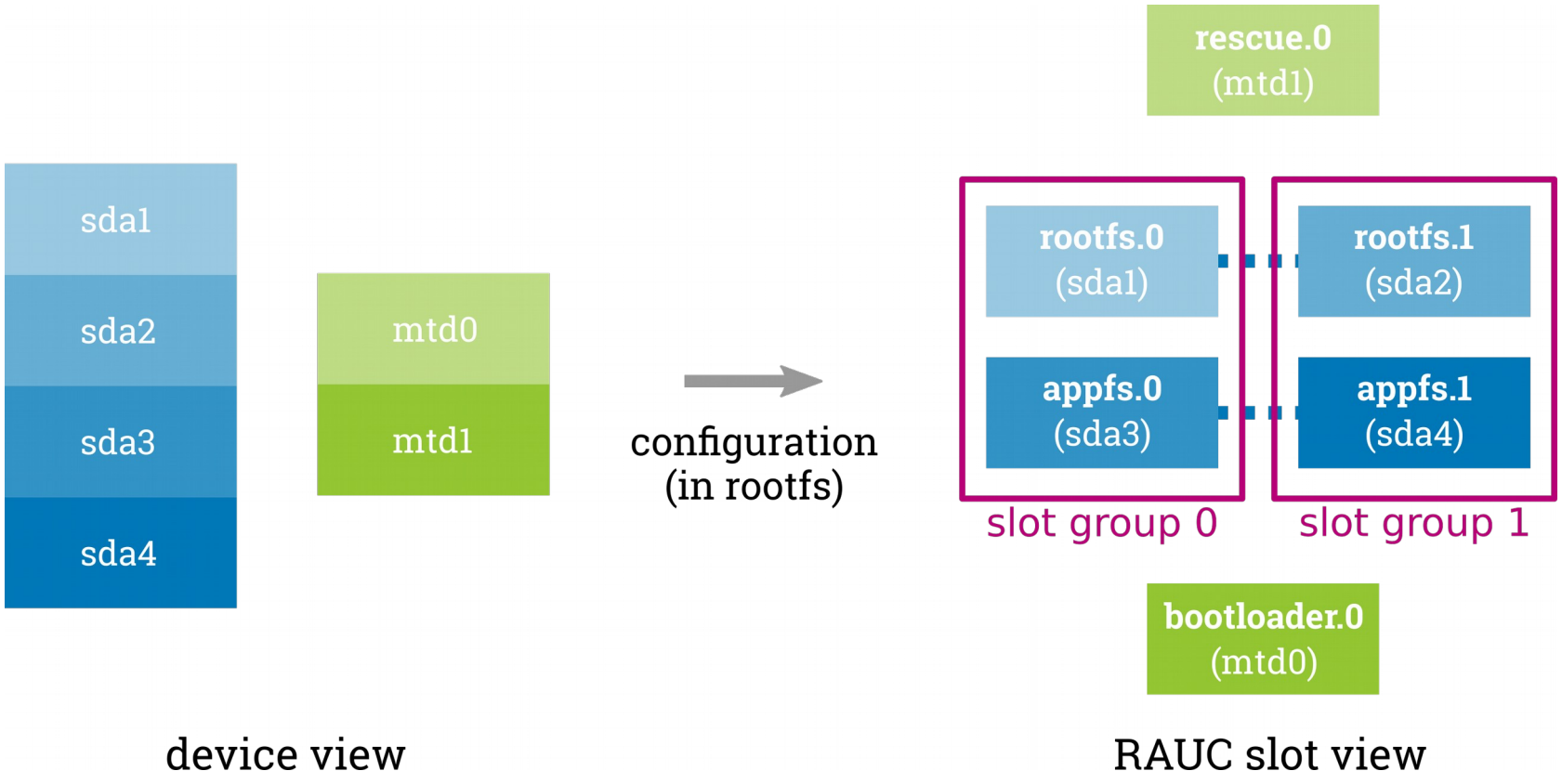
RAUC – Code & Service Architecture



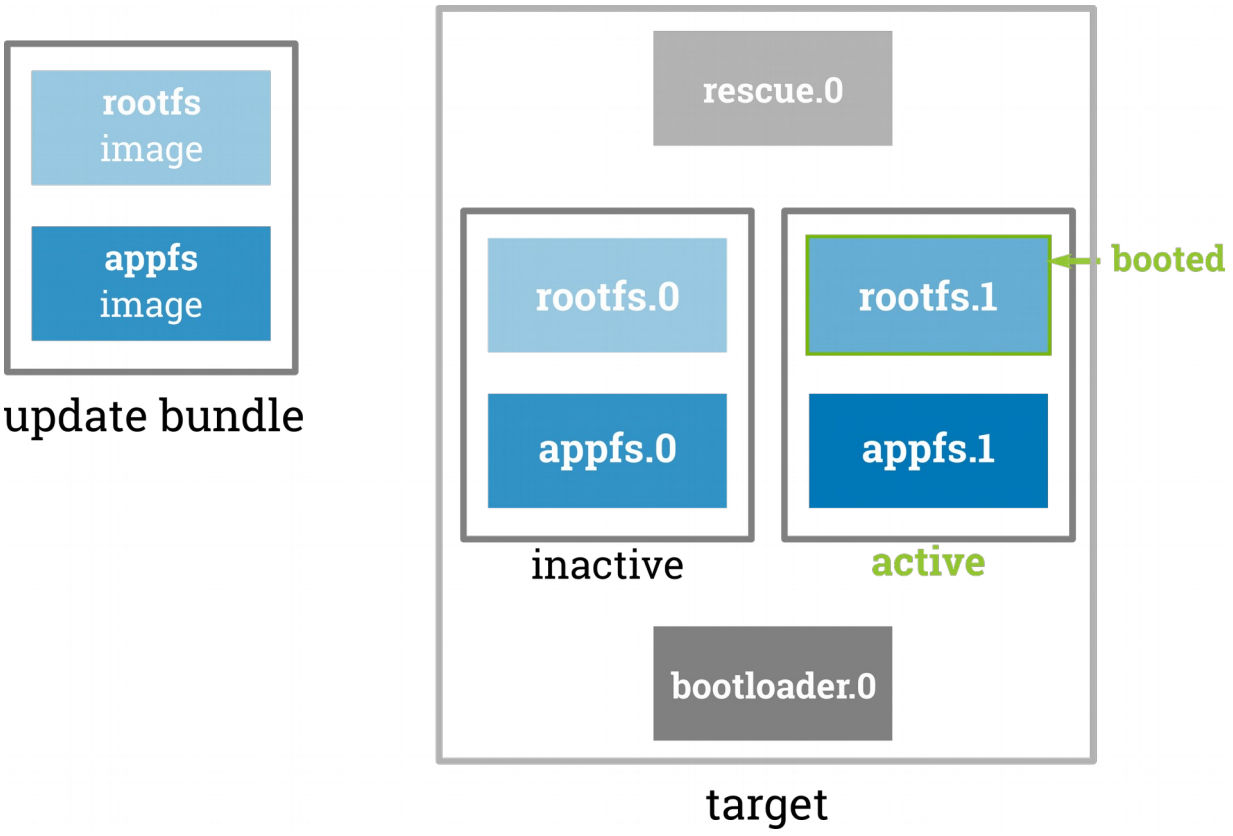
Service architecture

- C, automake
- Utility library → GLib
- D-Bus (optional)
- Subprocess calls for robustness

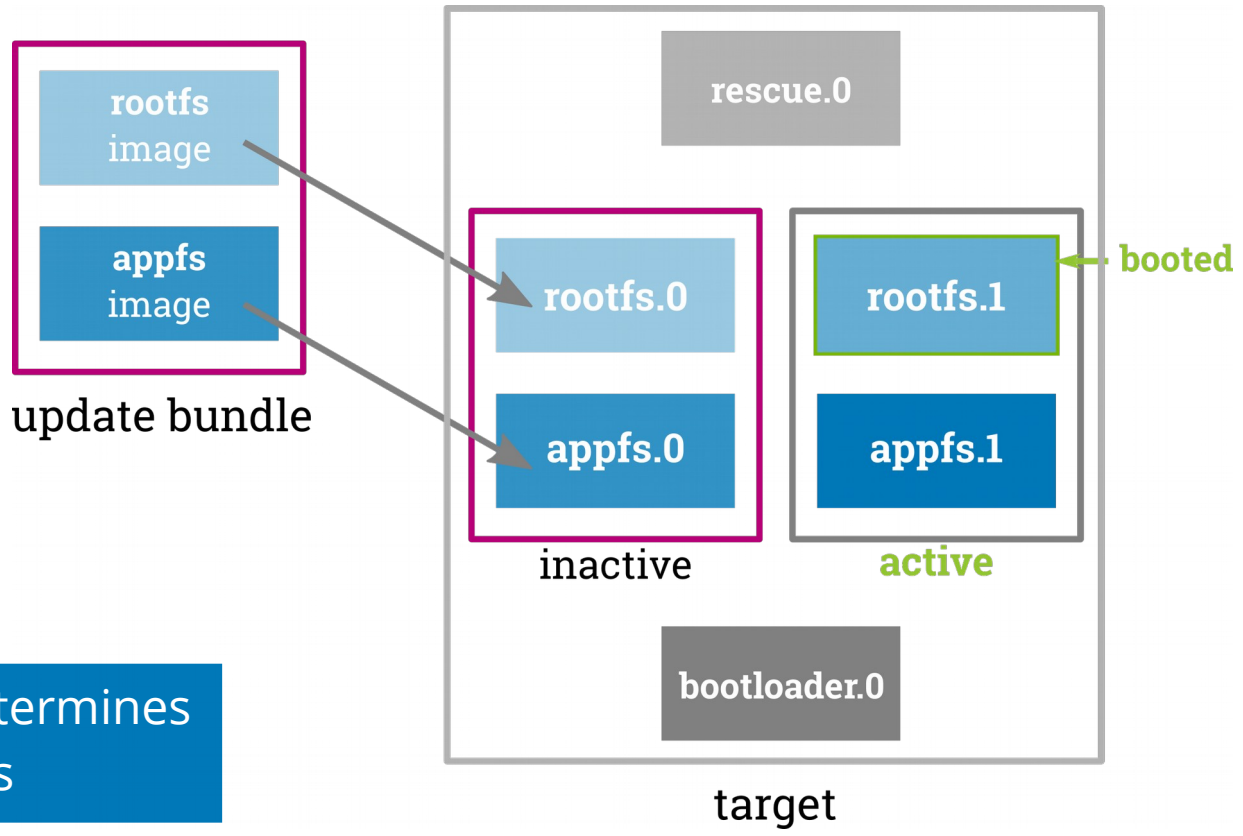
Slots – Configuring Redundancy Setup



RAUC – Slot Handling



RAUC – Slot Handling



Service determines target slots

Booted Slot Detection

- Preferably explicit

```
… console=ttyS0,115200 rauc.slot=system0 root=…
```

→ must match `bootname`

added by bootloader

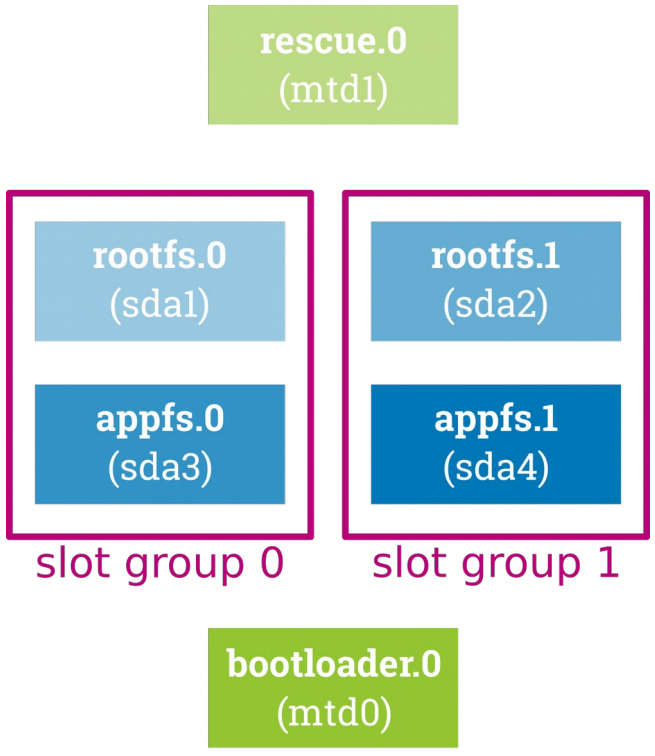
- Alternatively via root=

```
… console=ttyS0,115200 root=UUID=1b77ad0b-… …
```

→ must be detectable in mounted file systems

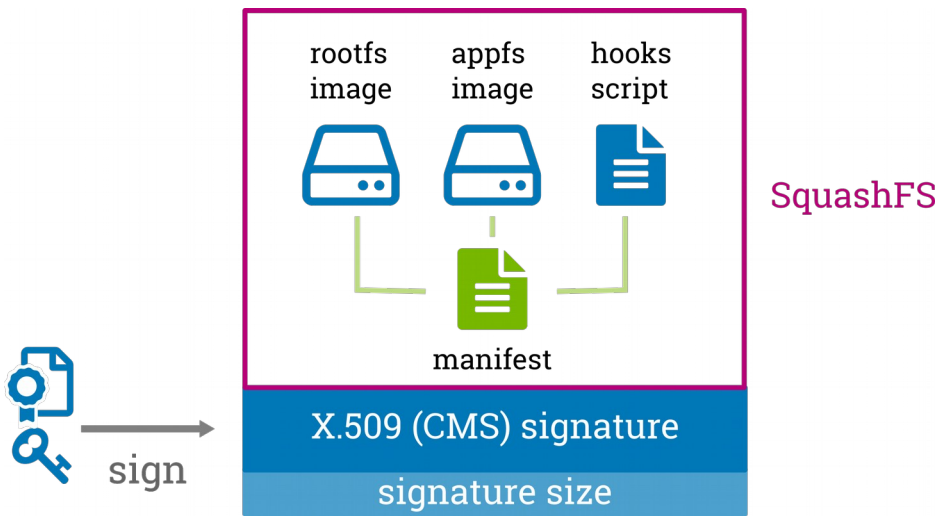
Introspection

```
$ rauc status
[...]
=== Slot States ===
  [bootloader.0] (/dev/mtd0, raw, inactive)

○ [rootfs.0] (/dev/sda1, raw, booted)
        bootname: system1
        boot status: good
    [appfs.0] (/dev/sda3, raw, active)

● [rootfs.1] (/dev/sda2, raw, inactive)
        bootname: system0
        boot status: good
    [appfs.1] (/dev/sda4, raw, inactive)
```



RAUC slot view

RAUC – Update Bundle Format



- SquashFS
  - Mountable!
  - Compression
- Signature (CMS)
  - Standard X.509 PKI
  - Verification on Target

A bundle describes the **intended target state** of the system

Manifest – System Configuration

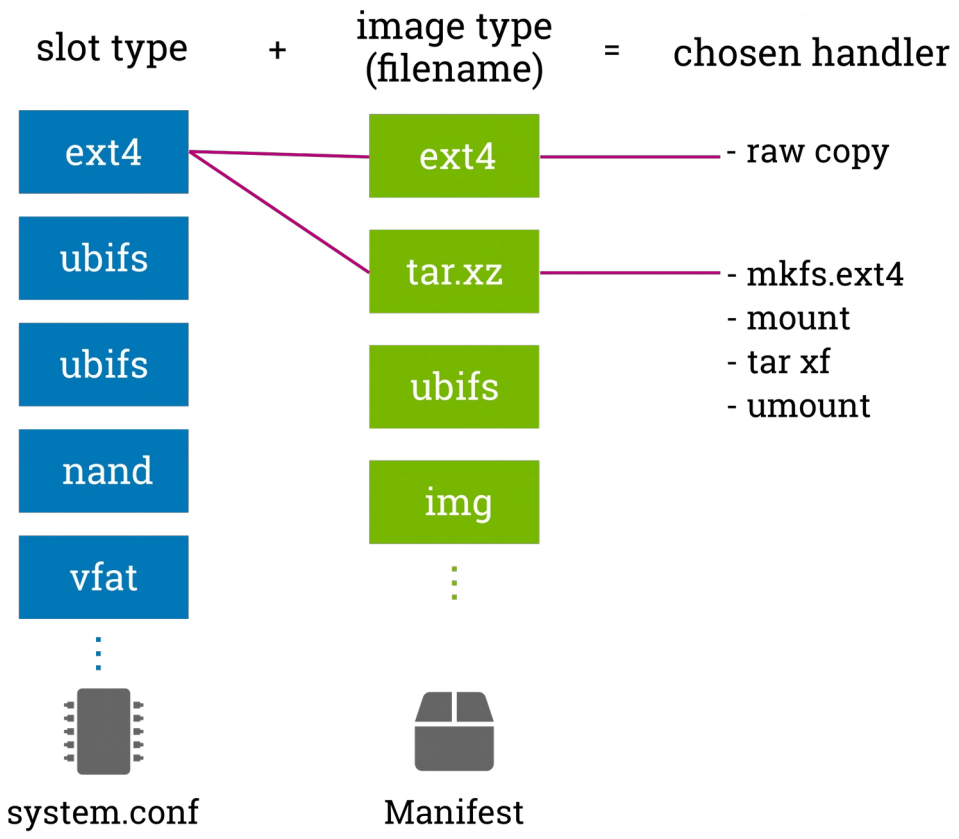
```
[update]
compatible=MyProduct2000
version=2019.10-4
build=20190228134503

[image.rootfs]
filename=rootfs.ext4
size=419430400
sha256=b14c1457dc1046...

[...]
```

Bundle manifest

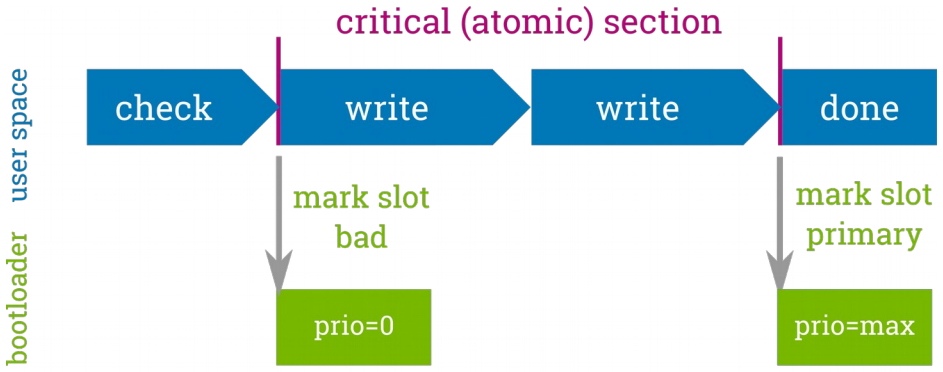
```
[system]
compatible=MyProduct2000
bootloader=barebox

[keyring]
path=/etc/rauc/keyring.pem

[slot.rootfs.0]
device=/dev/sda1
type=ext4
bootname=system0

[slot.rootfs.1]
device=/dev/sda2
type=ext4
bootname=system1

[...]
```

system configuration

HOW to install – Update Handler



RAUC Bootloader Interaction



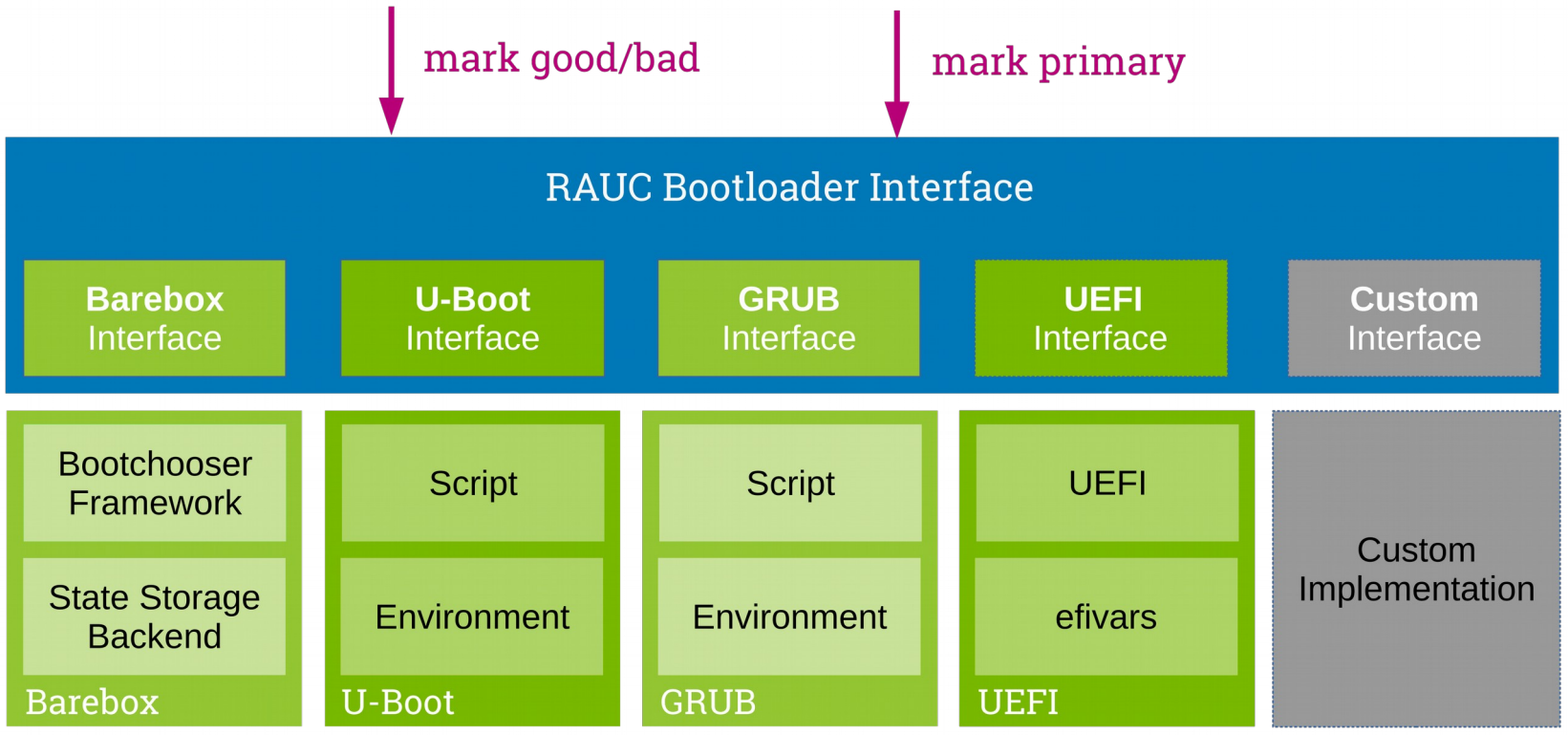
- Installation

  (Atomic action)

- Boot acknowledge

  (fallback handling)

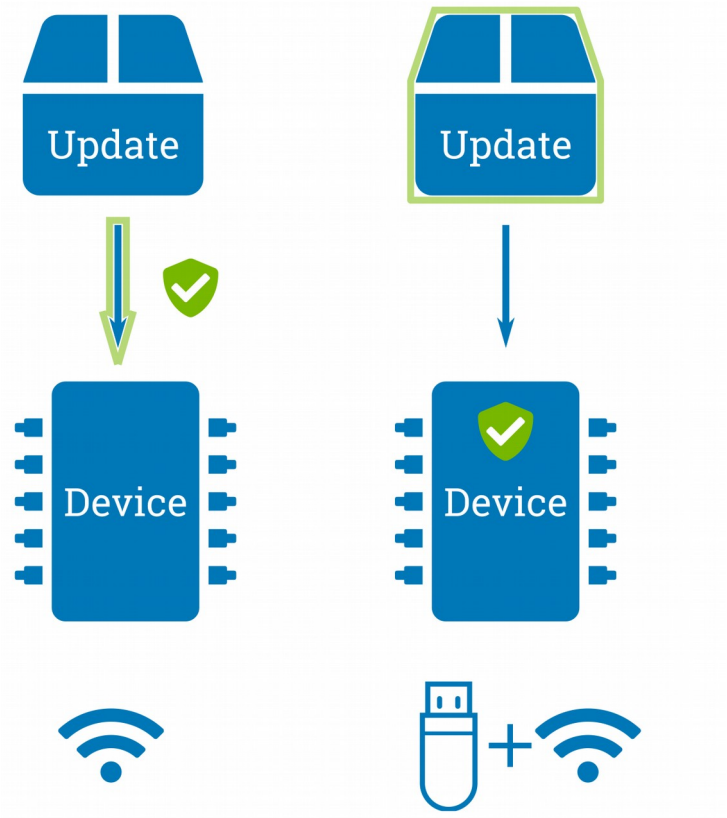
RAUC – Bootloader Interface



Customizing The Update

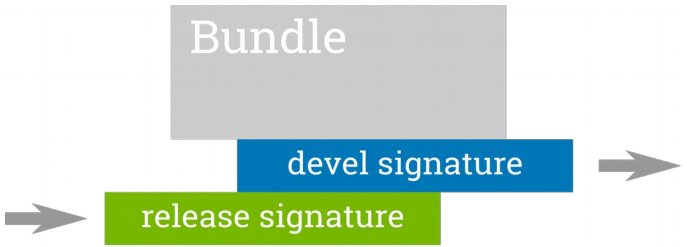
- System Configuration
- Hooks
  - In update bundle
- Handlers
  - In system
- Full custom handler
  - Only use signature verification

RAUC – Authentication

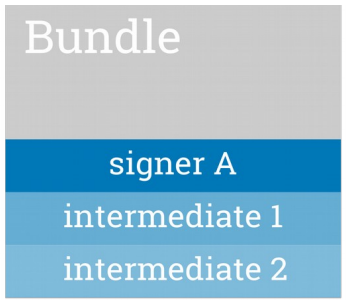


- RAUC host tool signs Bundle
- Verification on target
- OpenSSL 1.x
- X.509 crypto / CMS
- Self-signed to full PKI
- Key revocation and replacement

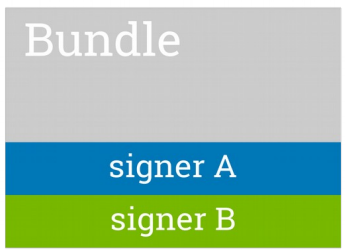
RAUC Signing Features– X.509 PKI



resigning



intermediate certificates in Bundle

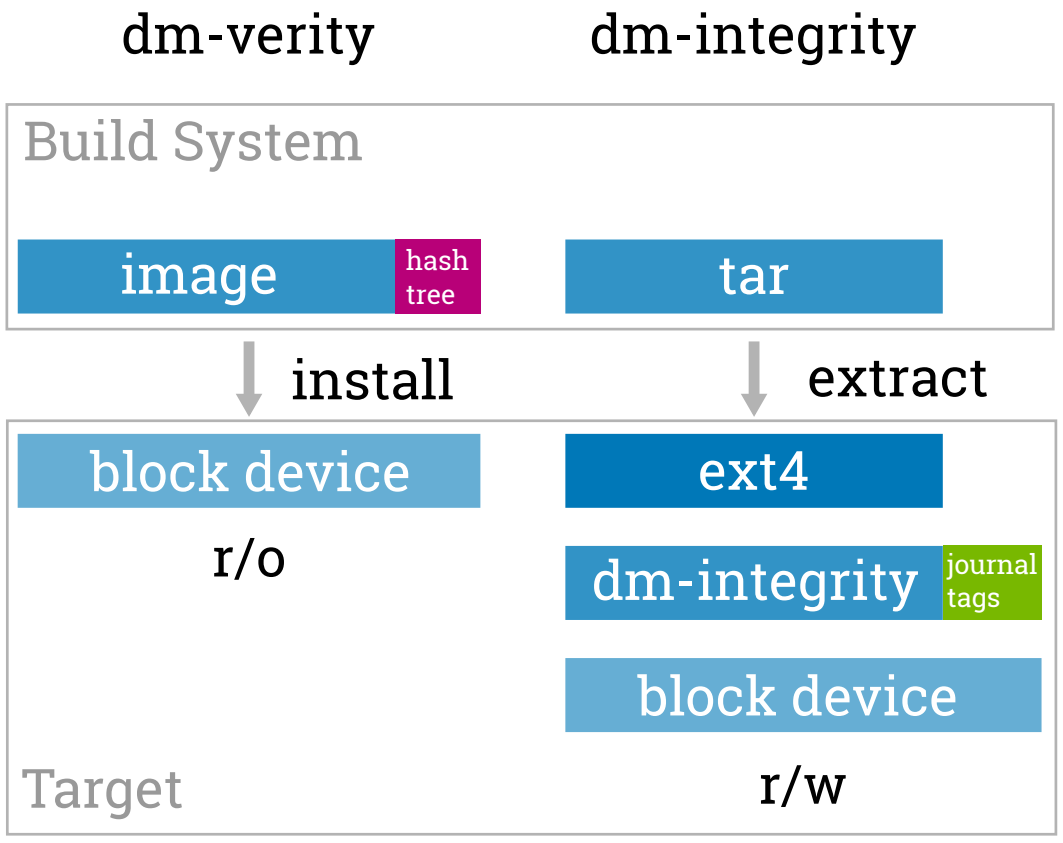


multiple signer

HSM (PKCS#11) support

RAUC – Advanced Topics

RAUC And Verified Boot



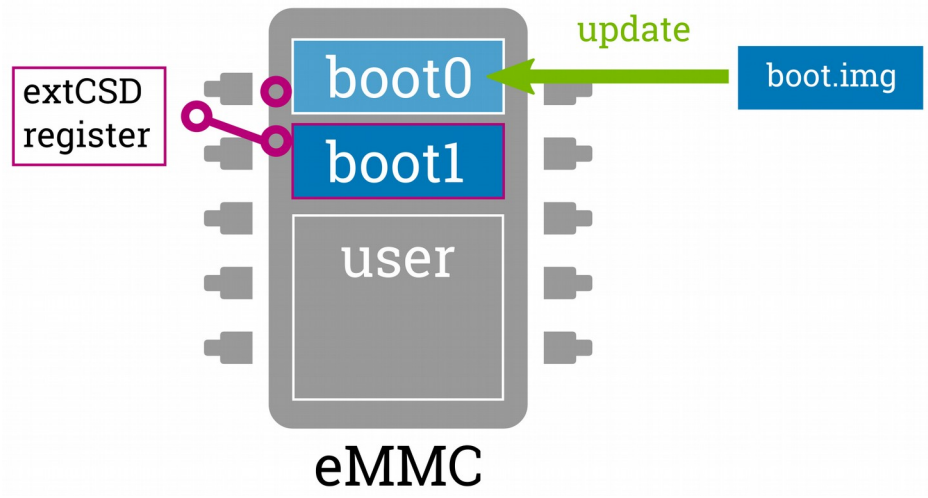
Built-In Bootloader Updates

- Single point of failure

  → Critical Component
- No Fallback, but atomicity!
- MBR
- If supported by ROM loader / storage:
  - eMMC
  - NAND (i.MX6)

```
...
[bootloader.0]
type=boot-..
device=/dev/device
...
```

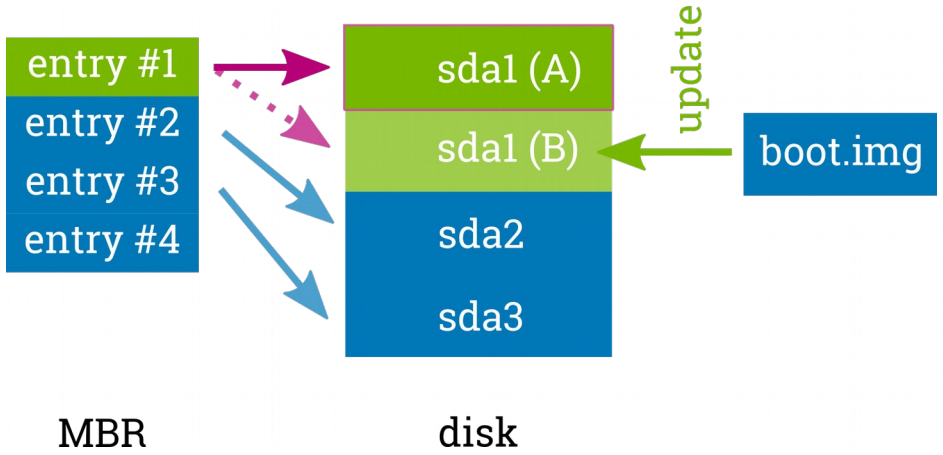
Atomic Bootloader Updates – eMMC



- built-in dual boot partitions
- extCSD register selects active one

```
...
[bootloader.0]
type=boot-emmc
device=/dev/mmcblk0
...
```

Atomic Bootloader Updates – MBR



- ROM loader boots from first MBR partition
- Switch between redundant partition regions

```
...
[bootloader.0]
type=boot-mbr-switch
device=/dev/sda1
...
```

Streaming and Delta Updates – casync

- Image updates over Network
  - Too large (slow connection)
  - Temporary storage required

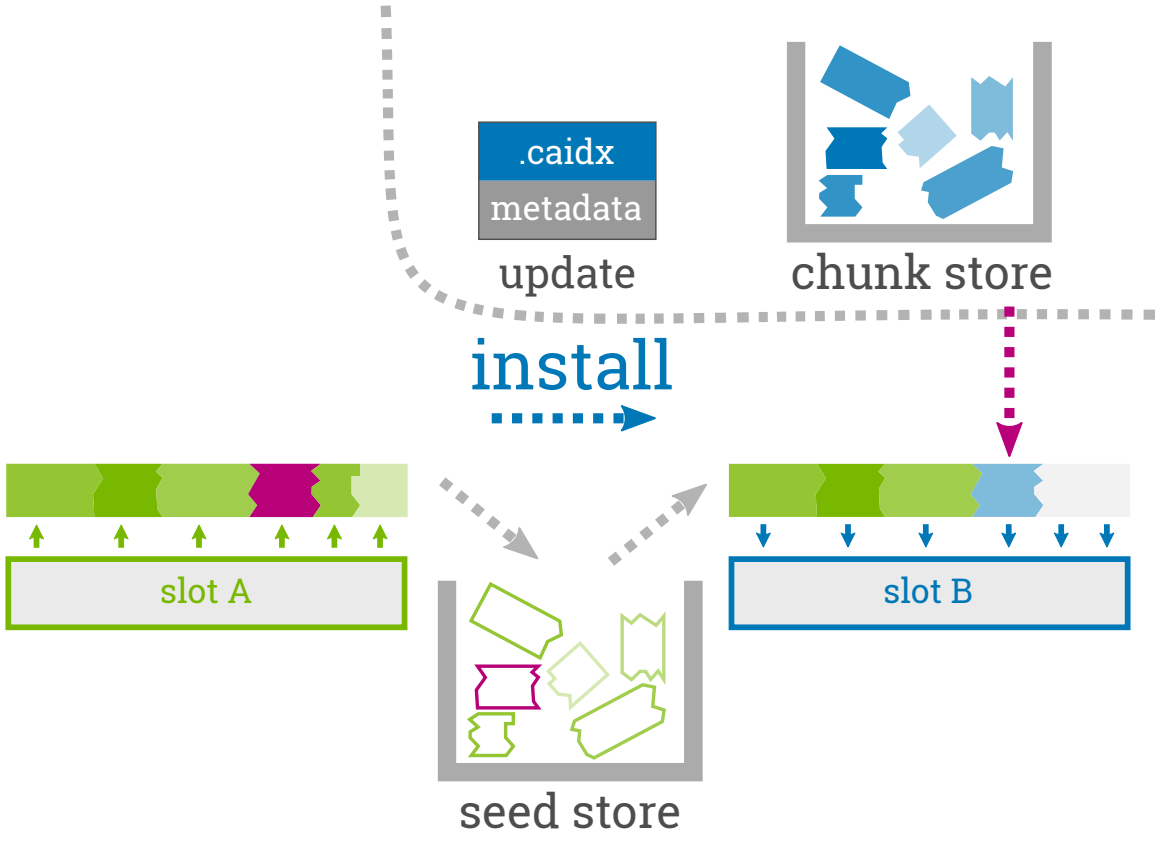
  → delta updates

  → avoid reinventing the wheel

“casync (content-addressable synchronisation) is a Linux software utility designed to distribute frequently-updated file system images over the Internet.“

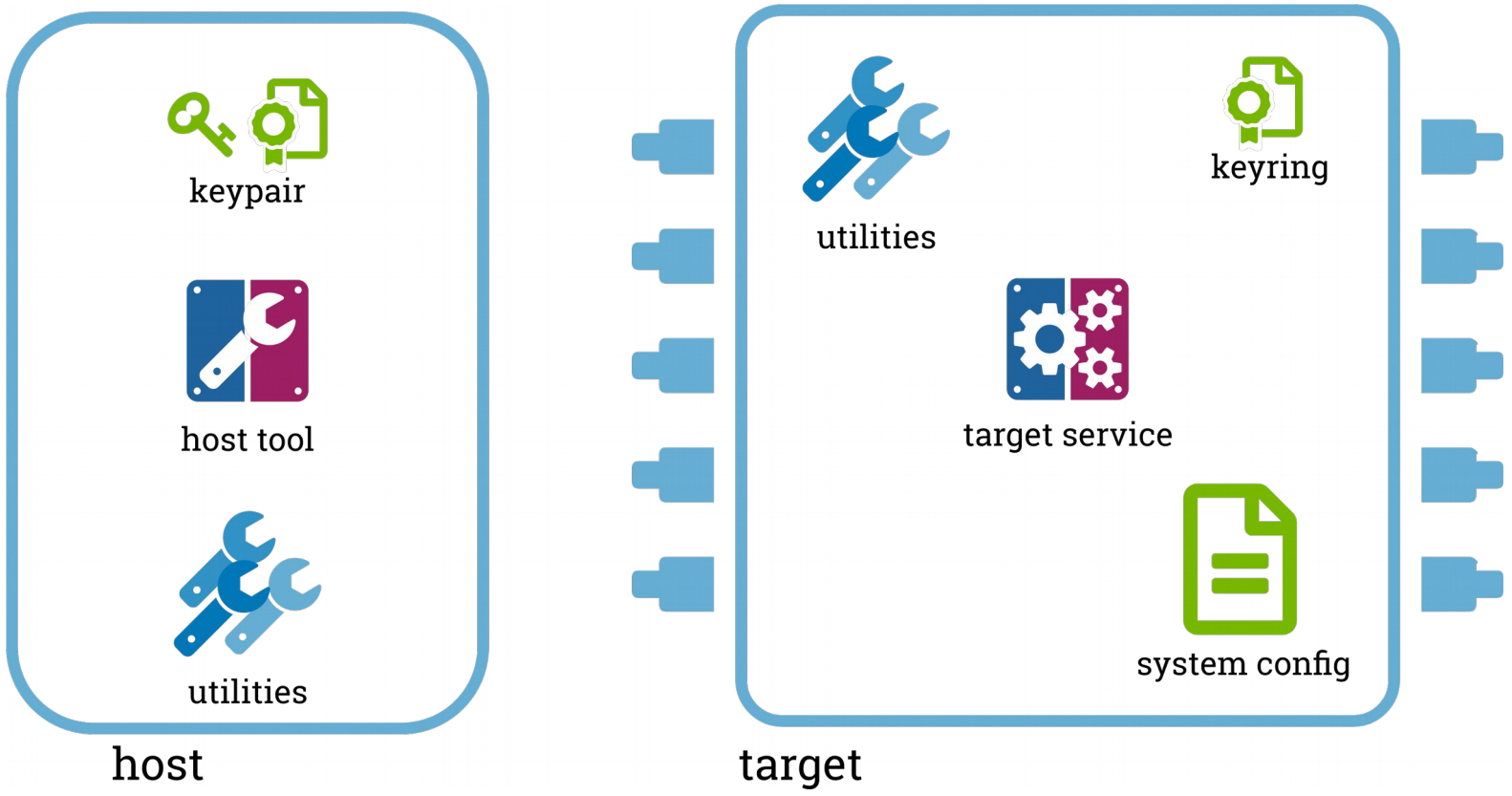
[Wikipedia]

casync – RAUC


.caidx
metadata
update
chunk store
install
slot A
slot B
seed store

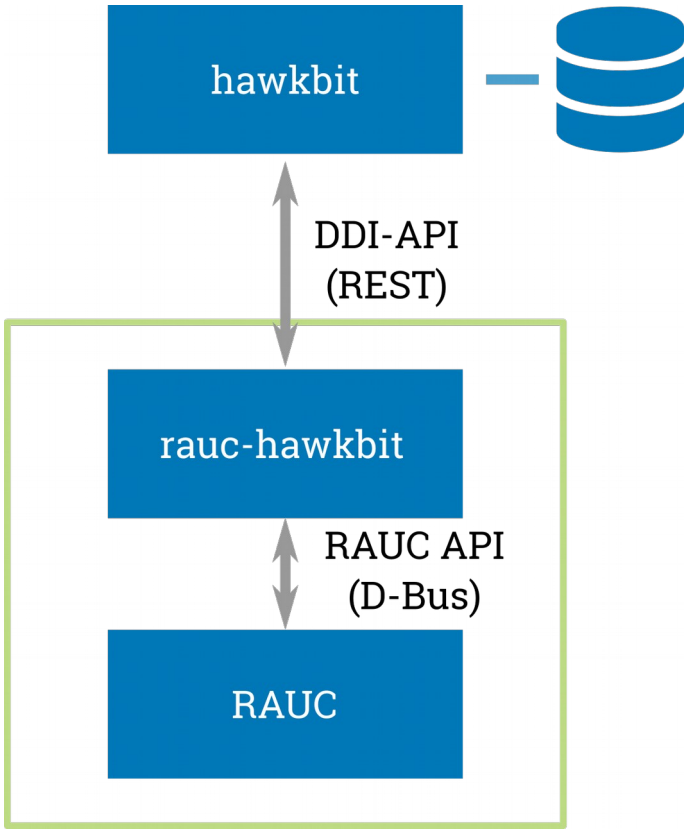
Usage and Integration

Integration – Required Components



Integration & Ecosystem

- Linux build system integration
  - Yocto (meta-rauc)
  - PTXdist
  - Buildroot
- Into Application: D-Bus
- Example projects like rauc-hawkbit



Thank you!

Questions?

References

RAUC system update documentation:

https://rauc.readthedocs.io/en/latest/

RAUC on GitHub:

https://github.com/rauc/rauc

meta-rauc:

https://github.com/rauc/meta-rauc

casync:

https://github.com/systemd/casync